Providing Easy Access to Distributed Medical Data

Bill Lord, Mark Tucker, Aninda DasGupta, and Mike Shneier
Philips Laboratories, Briarcliff Manor, New York

Many hospitals are fragmented along departmental boundaries, leading to islands of information about patients. This makes data integration difficult, and therefore can increase hospital costs and reduce patient care. This paper presents an architecture to provide uniform and transparent access to computerized data and functions available in this kind of heterogeneous computer environment.

INTRODUCTION

This paper presents an architecture to assist in accessing data and executing functions in a heterogeneous and fragmented information-gathering environment such as a hospital. The work is a component of a project undertaken to address problems that arise when medical practitioners need to access information from diverse sources[5]. A number of potential applications are envisaged, for example, reviewing a patient's record before a visit, preparation for interventions or surgery, producing diagnostic reports following a visit or procedure, and tele-consultations. Our project provides the information and tools needed to generate reports and review current patient records. A prototype application has been developed in the domain of cardiology.

Many of our goals are shared by other groups [7, 8, 12]. For example, the Hermes workstation project at Erasmus University in the Netherlands tries to integrate existing hardware and software into a physician's workstation [8, 9] without requiring substantial reprogramming. Our Data Server is a more general version of the Data Translator, Command Generator, and Network Facilitator modules that they have implemented, but the goals of hiding the details of communications protocols, the location of data, and the command language needed to retrieve the data are the same. Marrs et. al. at Washington University School of Medicine in St. Louis have implemented a system that integrates data from distributed heterogenous systems overnight, providing a global, unified view of this data by duplicating it on a central server[7]. Our Data Server provides the same global, unified view of the data, but without creating a new database.

The Current Dilemma in Medical Informatics

A typical hospital is made up of islands of information. Figure 1 shows a representative subset of a hospital's collection of computer systems. These systems are usually built around departments in a hospital, with each department storing the information it collects in its own computer system. Without a network between these systems, data from one department's computer system is often printed to paper, carried or faxed to another department, and then either scanned, re-entered manually or kept as paper. In fact, most patient records are still paper based even though almost 40 percent of their records were originally computer-generated [2].

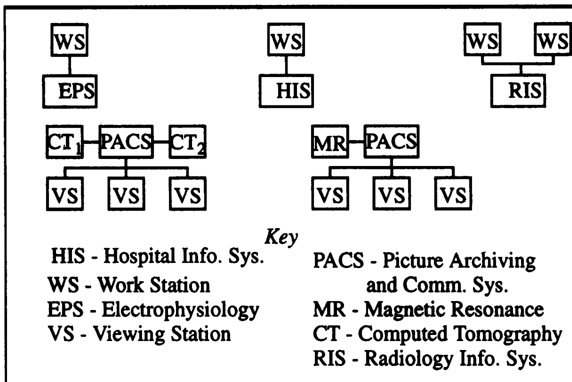

Figure 1. Islands of Information.

Simply networking different departments together is not sufficient to provide access between departments. There is still the potential problem of each system having a different operating system and/or database query language since each department typically chooses the computer system that best suits its needs. To obtain information from the islands requires knowledge specific to each system that will be accessed. The premise behind our work is that end users and application programmers want to be insulated from the details of data access. They want to think only in terms of available functionality and data models. They do not want to have to think in terms of protocols, data and function location, and data formats. Thus, we have developed methods to hide these system-specific details. Our solution focuses on information access techniques that enable site-independence. It is designed to allow the system to be introduced at a new location without requiring additional programming.

Possible Solutions

A common approach to integrating information from a large number of sources is to use a hub concept, in which a central location serves to integrate and often to store the information. In most of these systems (e.g., [3, 6, 7]), the information sources are expected to transmit information to the hub at scheduled times or as it becomes available. The hub integrates the information and responds to queries from users or other systems. It can also redistribute information. An advantage of the hub concept is that it is relatively easy to implement and may simplify the problem of maintaining consistency in the data. Disadvantages are that the hub has to store copies of all information, that network traffic is increased, and that the hub can potentially become a bottleneck.

The approach that we have taken is based on what we call a Data Server. The Data Server presents a single data model for the collection of a hospital's data. It functions by having knowledge of where and how requested data are stored, not by storing all data in a central repository. The data server accepts requests and forwards them to the appropriate system in the proper format for that system.

What the hub systems and the Data Server have in common is that they both hide from the user the idiosyncrasies of the information sources. While a hub system stores data, the Data Server stores information on how and from where to retrieve and manipulate data. In both cases, a connection to the equipment must be built. If there is no well-defined query language (such as SQL) and the system does not support HL7 [4], then a custom data interface must be written and installed on the system.

Our initial goal for the Data Server was to provide database access to all hospital data without needing knowledge of where the data actually reside. In the same way that we want to provide uniform access to data, we also want to be able to access the collection of available functions (such as image processing, for example) from all the hospital's systems in a uniform way.

The purpose of this paper is to discuss the architecture and various configurations for our Data Server. To set the stage for this, we give a brief overview of what a simple Data Server configuration does and how it works.

OVERVIEW OF THE DATA SERVER

The Data Server provides uniform and transparent access to electronically available information in a hospital. The Data Server unifies all communication needs and provides common entry and exit points for exchanges between different systems. The Data Server is the only system that needs to know all the communication protocols and database query languages used in hospitals, clinics, or satellite offices. When it receives a request, the Data Server must determine where to find the necessary information and must also know how to ask the owner of the information to retrieve it. It often has to break a request into a number of simpler requests, each of which is sent to a different information source using the language expected by that source. Information may be returned in any order and at any time, so the Data Server has to reconstruct the answer to the request, remembering which pieces belong to which requester, and send the answer using the proper protocol back to the user. Because many systems that supply information are dedicated to information-gathering, they may not be able to accept or respond to requests except at specified times. Thus, the Data Server must handle queues both for requests and for responses. The architecture allows for more than one Data Server to operate at the same time, so bottlenecks can be avoided.

Data Dictionaries

Data Dictionaries are a key resource of the Data Server. Data Dictionaries give the appearance, and are accessed as if they were simple tables of a relational database. However, instead of containing the actual data sought, they contain the precise descriptions needed to retrieve the data requested from its source. Entries include information about the device on which the requested information is stored, how to format a query for that device, how to interpret a response from the device, and if needed, how to expand the single query into a collection of multiple queries. This latter information allows for both simple and complex queries to appear to the requesting application program as simple requests. For example, a request from an admissions workstation for the description of a patient's last hospital visit will be expanded in the Data Server into requests to one or more devices for a date, location, and problem for that visit. These results will be collected on the Data Server, formatted into a single response and sent back to the requesting workstation. Specifying how to interpret results from various systems allows for the data server to run functions on collected data to adjust for mismatched schemas before sending the results back to the requesting application. An example of this would be an entry in the Data Dictionary that specifies that a function to convert pounds to kilograms be run whenever a patient's weight is obtained from the hospital's Hospital Information System (HIS).

We have developed a graphical user interface for specifying the layout and contents of the data dictionaries. This specification will typically be done by a system administrator who is knowledgable about the data schemas of each of the individual systems that is to be integrated into the virtual database of the Data Server. The Data Dictionary can be set up as a single global view of the union of all the data from all the systems in the hospital, or it can be set up to have a collection of multiple views of either all the data or various subsets of it.

Data Server Functions. We model how the Data Server works at the function level after work in the object oriented systems community [11]. This model requires all systems on a network to support a standard way of registering their addresses and what they can offer to others, as well as an authentication and access control mechanism. The Data Server becomes a distributed services broker, acting as an intermediary between clients (processes needing services) and service providers (processes controlling access to services). Services may include much more than information retrieval or storage. The client does not need to know where on the network the services are provided, or how the desired results are generated. Similarly, the server does not need to know for which clients it is providing the information (assuming that the authentication and security aspects are properly dealt with). Two evolving standards activities that address this architecture are the ISO/CCITT Open Distributed Processing (ODP) effort and the Object Management Group's Common Object Request Broker (CORBA) [10, 11].

Data Server Components

The Data Server consists of three kinds of components: client adapters, a request processor, and server adapters.

Client Adapter. The client adapter accepts incoming requests in a particular protocol from a client and calls an interface to the request processor. The role of the adapter is to convert system-specific protocols to and from a standard format. A client, in our terminology, is any process that requests information from a Data Server. Client adapters can be built for any known protocol. Currently we have client adapters for ACR-NEMA 2.0, HL7, and an internal protocol [1, 4]. These adapters can all run within a single process, or run as separate processes. More than one of each kind of adapter can run simultaneously. It is possible to have one adapter per client. A client adapter can run on the same system as the request processor portion of the Data Server, or separately.

Request Processor. The request processor supports different kinds of requests. Our discussion focuses on one class of request, data queries. The request processor for data queries consists of a request parser that creates request objects, and a request engine that executes them. In the request parser, queries are parsed (broken down) into atomic database calls. The collection of atomic calls forms the main content of a request object. These atomic database calls consist only of requests that are simple enough to be handled even by the least sophisticated of databases. For example, a query to find all of the reports written for all catheterization procedures performed on patient Jane Doe, would be broken in to three simple requests: 1. Find *Patient ID* given *Patient Name* of Jane Doe, 2. Find all the *Procedure IDs* for catheterization procedures performed with the *Patient ID* returned from the first simple request, and 3. Find all the *Reports* associated with all the *Procedure IDs* found in the second simple request. When sent to the request engine the atomic calls consist of three parameters: a relation name, an attribute name to search in, and the attribute's name from which to obtain the result. When the request is ready to be processed, a fourth parameter is added: the value to search for. If the original query to the request processor ends up as more than one atomic call, then each atomic call may have to go to a different system.

The request engine sequentially takes each atomic call and determines which process on the network can answer it. We refer to these processes as "servers," which usually, but not necessarily, run on a system other than the Data Server. Examples of typical servers are HIS and RIS systems. The atomic call is passed on to the server adapter for the server that holds the desired information. After passing on the atomic call, the request engine is free to service another request object. Further processing of a request object can be resumed after the server replies. After an atomic call is completed and the request engine is re-entered, the results of the atomic call will be used as the search values for the next atomic call

Server Adapter. A server adapter fulfils two purposes. First, it determines if the system to which the request is destined can handle multiple requests or not. Second, when the request is ready to send, the server adapter translates the parameters given by the atomic call into a system-specific request needed to obtain the desired information. If the information server can handle multiple requests, the request is sent out. If not, the request is put on a queue for that system. Because the server adapter may be required to maintain a queue for the server, only one server adapter can exist for each server. Server adapters can run one per process, or grouped into one or more processes.

DATA SERVER ARCHITECTURES

There is a lot of flexibility in the architecture for putting together the components of the Data Server. The simplest architecture has all three Data Server components running in the same process. This configuration is shown in Figure 2 and is based on the representative systems of a hospital presented in Figure 1.

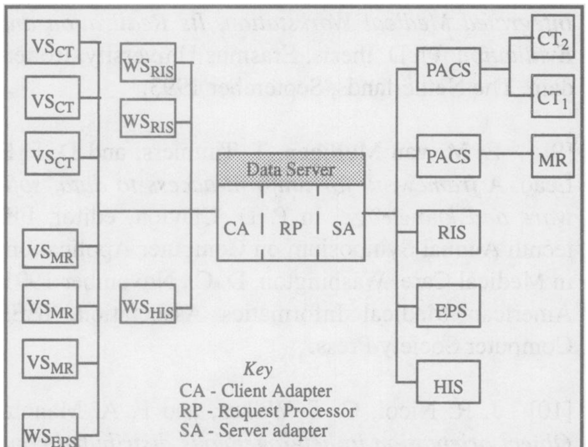

Figure 2. Logical View of Network With Data Server.

Figure 2 shows a logical view of how the hospital would work with our Data Server. All the viewing stations and the workstations would get information they needed from the data sources via the data server. Thus allowing for location hiding.

However, this configuration can lead to a bottleneck. We address this potential problem in two ways. First, most requests made to the Data Server are of low bandwidth while the answers may require high bandwidth. For example, a request for a patient's cine loop from a recent catheterization could be less than 100 bytes long. The data set requested could be on the order of tens of megabytes. Sending this from the archives system to the requester via the Data Server could take twice as long as sending it directly, and would tie up valuable resources. Instead the archives can either send the data set directly to the requestor, or send the requestor enough information about accessing the data that the requestor can then access it directly.

The second approach we have taken to alleviate the problem of a Data Server becoming a bottleneck is to have parts of the Data Server duplicated, and running in a distributed environment. One logical configuration is to use a client adapter and request processor for each cluster, and a server adapter for each information source. Each client adapter/request processor pair can run separately in its own address space, or they can co-exist in a process with the client. Similarly, a server adapter and server can run separately or within the same process. Figure 3 shows different configurations of the Data Server.

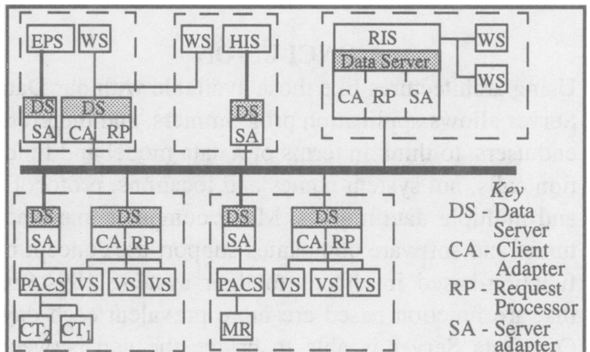

Figure 3. View of Network With Distributed and Redundant Components of the Data Server.

In this figure we see that the client adapter / request processor pair in the Electrophysiology Department runs as a separate process and is shared by the workstation associated with the hospital information system. The Radiology Information System has all three components of a Data Server compiled in with its code and all running as one process. The Magnetic Resonance cluster and the Computed Tomography cluster show typical configurations for cluster based topologies.

FUNCTION BASED INTERFACES

In the preceding sections we examined the use of the Data Server to provide uniform access to data. By expanding on the idea, the Data Server can become a means to uniformly access many of the other functions offered by the systems in a hospital, such as image processing, text processing, and drug interaction checking. In a manner similar to how it keeps knowledge of where data reside and the protocols required to retrieve them, the Data Server can also keep track of where available functions reside and how they can be executed.

In an ideal setup, each system on a network would advertise its capabilities and provide a standard method of accessing them. In this case the Data Server, if necessary at all, would only have to act as the "yellow pages." Because this ideal world does not exist yet, the Data Server is designed to give the appearance that it does.

Functions do not have to be a one-to-one mapping to the actual functions of the various systems on the hospital's network. For example a function on the request processor may advertise that it can supply annotated images, when in fact no system in the hospital supplies this capability directly. Instead the request processor has to retrieve the image in question from a PACS system, and then retrieve the dictation from another system and combine the two files together in the appropriate multimedia format for return to the requesting system.

CONCLUSION

Using architectures like those available with our Data Server allows application programmers, and therefore end users, to think in terms of a data model and function calls, not system names and locations, protocols, and multiple data models. Many computer manufacturers and software companies support the concept of function-based interfaces to their systems. Systems that are function based are more prevalent each day. Our Data Server is able to bridge the gap between older and current day systems, providing the benefits of seamless data integration and function specification.

References

[1] *ACR-NEMA Dig. Imaging and Comm. Stds. Ctee.*, ACR-NEMA 300-1985:Digital Imaging and Communications. 1985

[2] M. L. Cannavo. *Fitting radiology into the hospital of tomorrow.* Diagnostic Imaging, pages 131-133. April 1990.

[3] A. C. Curtis. *Multi-facility integration: An approach to synchronized electronic medical records in a decentralized health care system.* In Lun et al., editors, Proceedings of the Seventh World Congress on Medical Informatics (MEDINFO 92), Geneva, Switzerland, September 1992, pages 138-143. IFIP and IMIA, North-Holland.

[4] Health Level Seven, 900 Victor's Way Suite 122, Ann Arbor, MI 48108. *Health Level Seven: An Application Protocol For Electronic Data Exchange In Healthcare Environments.* Version 2.1.

[5] H. K. Huang, R. L. Areson. *Multimedia demands integrated databases.* Diagnostic Imaging. Vol. 15, #11. November 1993.

[6] S. B. Johnson, P. D. Clayton, D. Fink, et al. *Achievements in phase III of an integrated academic information management system.* In Lun et al., editors, Proceedings of the Seventh World Congress on Medical Informatics (MEDINFO 92), Geneva, Switzerland, September 1992, pages 117-123. IFIP and IMIA, North-Holland.

[7] K. A. Marrs, S. A. Steib, C. A. Abrams, and M. G. Kahn. *Unifying Heterogeneous Distributed Clinical Data in a Relational Database.* In Safran, editor, Seventeenth Annual Symposium on Computer Applications in Medical Care, Washington, D. C., November 1993. American Medical Informatics Association, McGraw-Hill. 1994. Pages 644-648.

[8] E. M. van Mulligen. *An Architecture for an Integrated Medical Workstation, Its Realization and Evaluation.* Ph.D. thesis, Erasmus University, Rotterdam, The Netherlands, September 1993.

[9] E. M. van Mulligen, T. Timmers, and D. F. F. Leao. *A framework for uniform access to data, software and knowledge.* In P. D. Clayton, editor, Fifteenth Annual Symposium on Computer Applications in Medical Care, Washington, D. C., November 1991. American Medical Informatics Association, IEEE Computer Society Press.

[10] J. R. Nicol, C. T. Wilkes, and F. A. Manola. *Object orientation in heterogeneous distributed computing systems.* IEEE Computer, 26(6):57-67, June 1993.

[11] Object Management Group, Framingham Corporate Center, 492 Old Connecticut Path, Framingham, MA 01701-4568. *The Common Object Request Broker: Architecture and Specification.* OMG Document Number 92.12.1, 1991.

[12] J. R. Scherrer, R. Baud, D. Hochstrasser, and O. Ratib. *An integrated hospital information system in Geneva.* M. D. Computing, 7(2):81-89, 1990.